2019 年度修士論文発表会

定理証明支援系 Coq によるデータ型汎用なプログラム運算

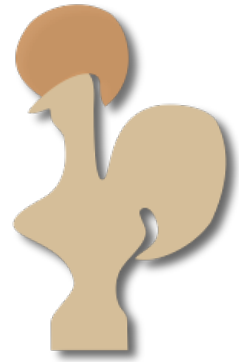
2020/02/12

村田 康佑

江本研究室

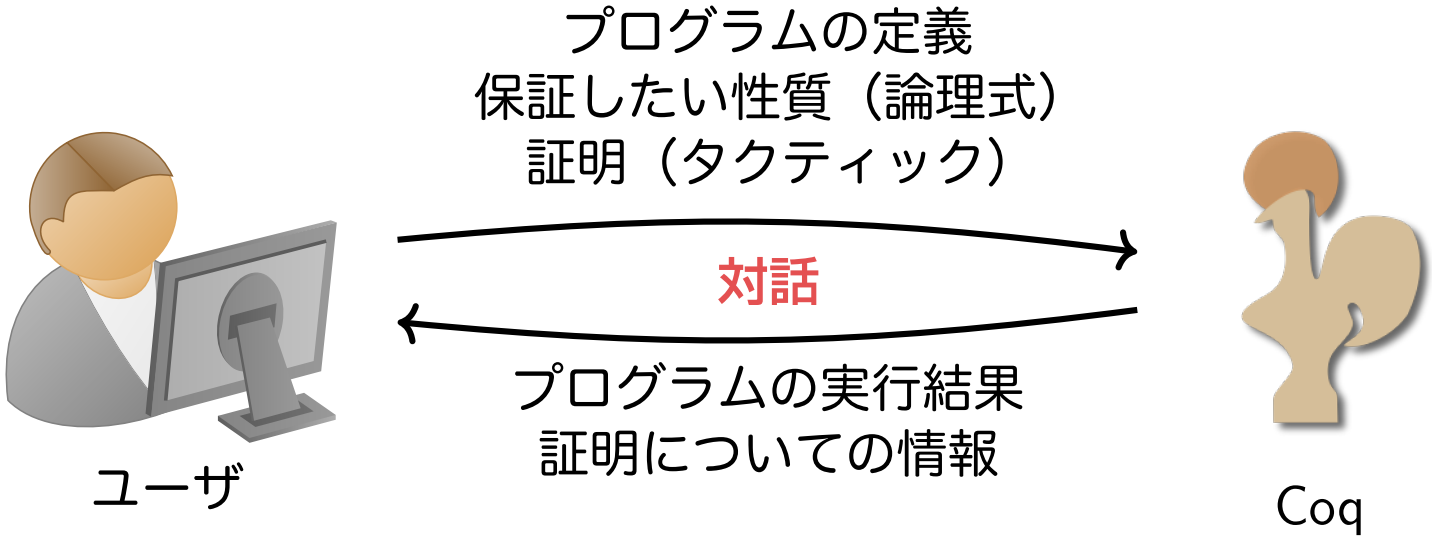
背景 1:
ソフトウェアの品質保証と Coq

- ソフトウェアの品質保証は重要
  - 単にバグがないことだけでなく，
    バグがないことの**保証**が求められる
- **定理証明支援系**を利用したアプローチが盛ん
  - **数学的証明**を利用してバグがないことを示す
    - C コンパイラの正当性検証 [Blazy+ '05] などに利用
- Coq: 定理証明支援系の一つ
  - 依存型理論に基づく証明を与える
  - 直観主義 ZF と同値な型理論に基づく



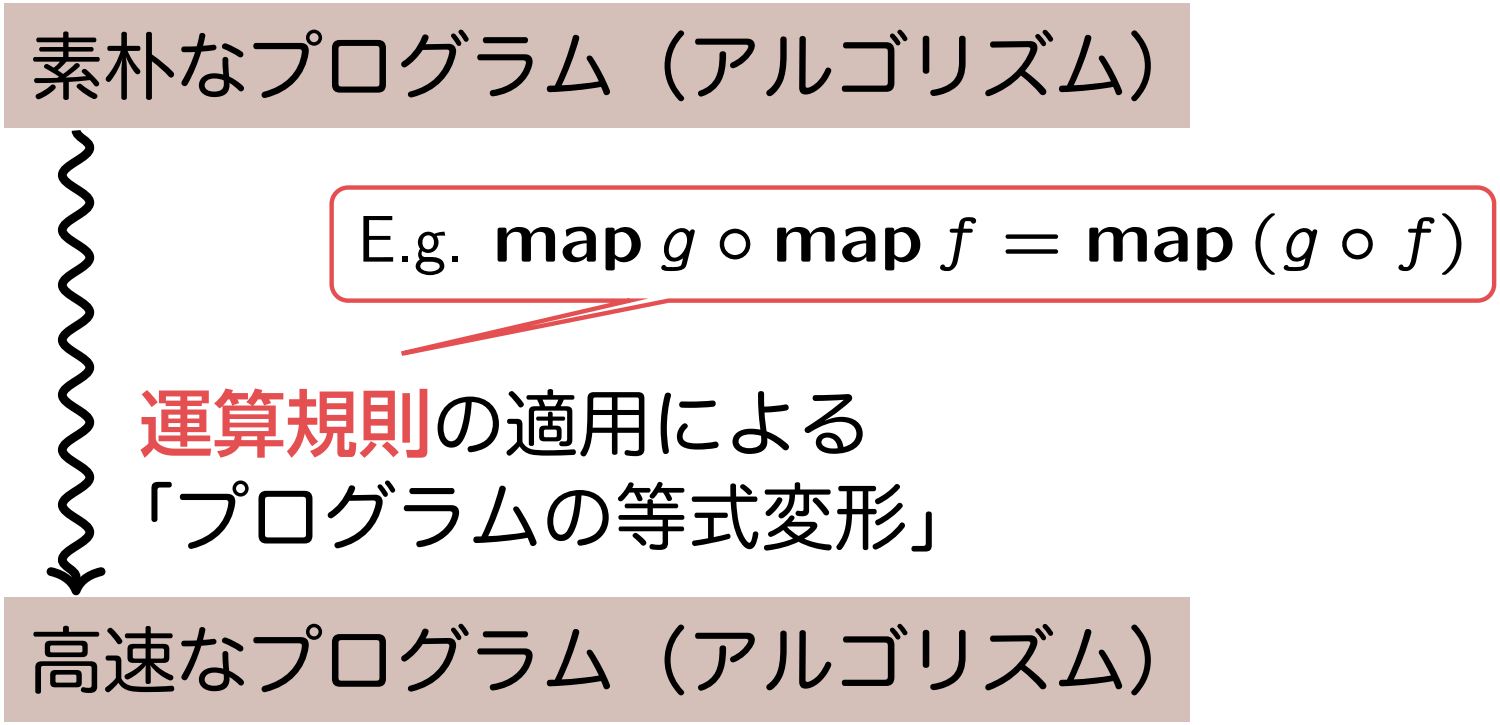
Coq: 対話的定理証明支援系

- ユーザの証明を**対話的**に検証・補助
- 中核に関数型言語 Gallina をもつ
  - プログラムの定義や実行が可能
  - プログラムの性質を論理式で記述し，証明



背景 2:
プログラム運算

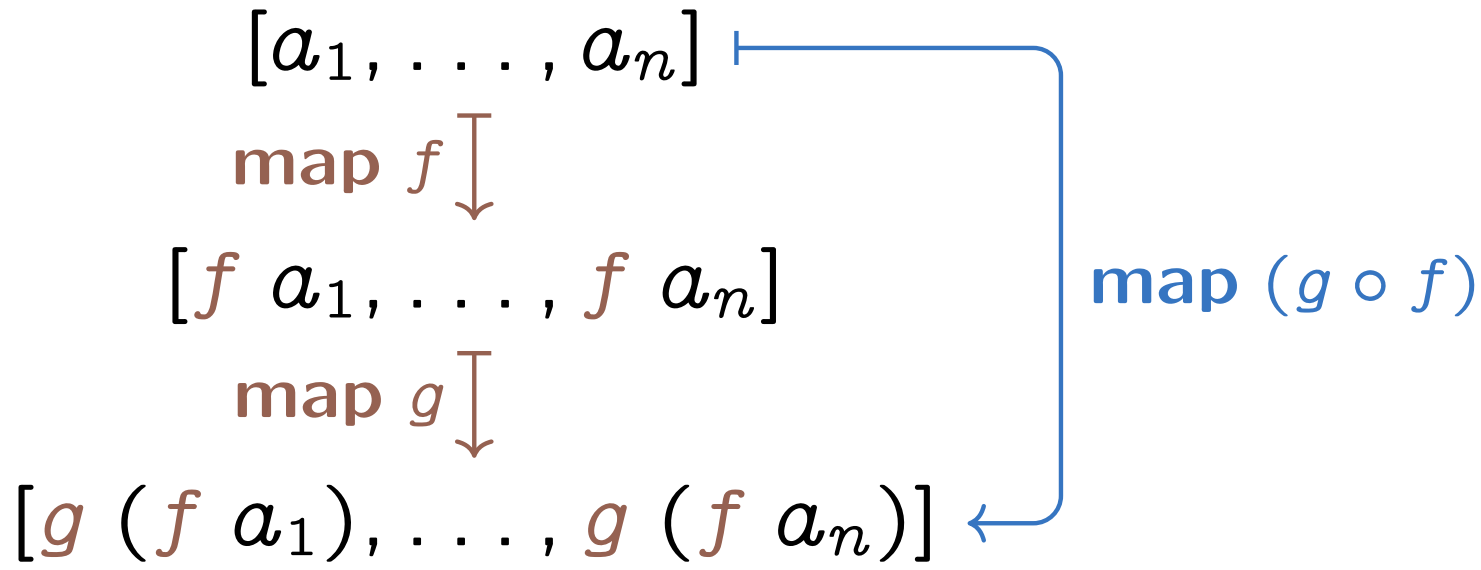
- 素朴なプログラムから高速なプログラムを導出
  - **運算規則**といわれるプログラムの間の**等式**を利用



運算規則の例:
map fusion law

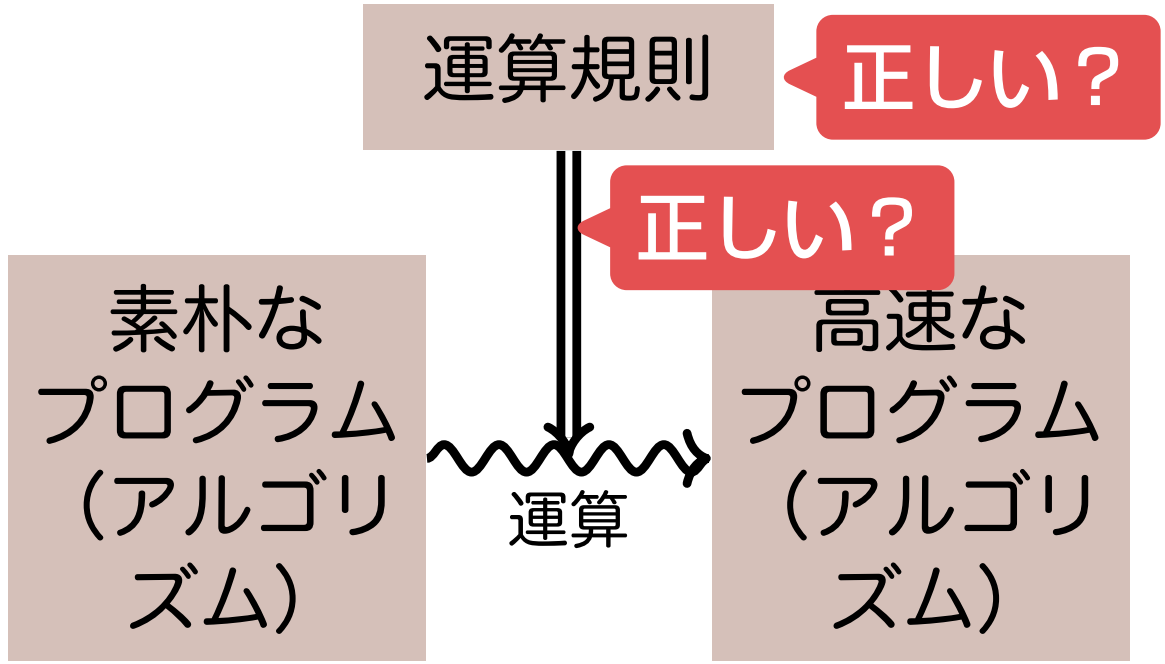
$$(\mathbf{map}\ g) \circ (\mathbf{map}\ f) = \mathbf{map}\ (g \circ f)$$

For Lists



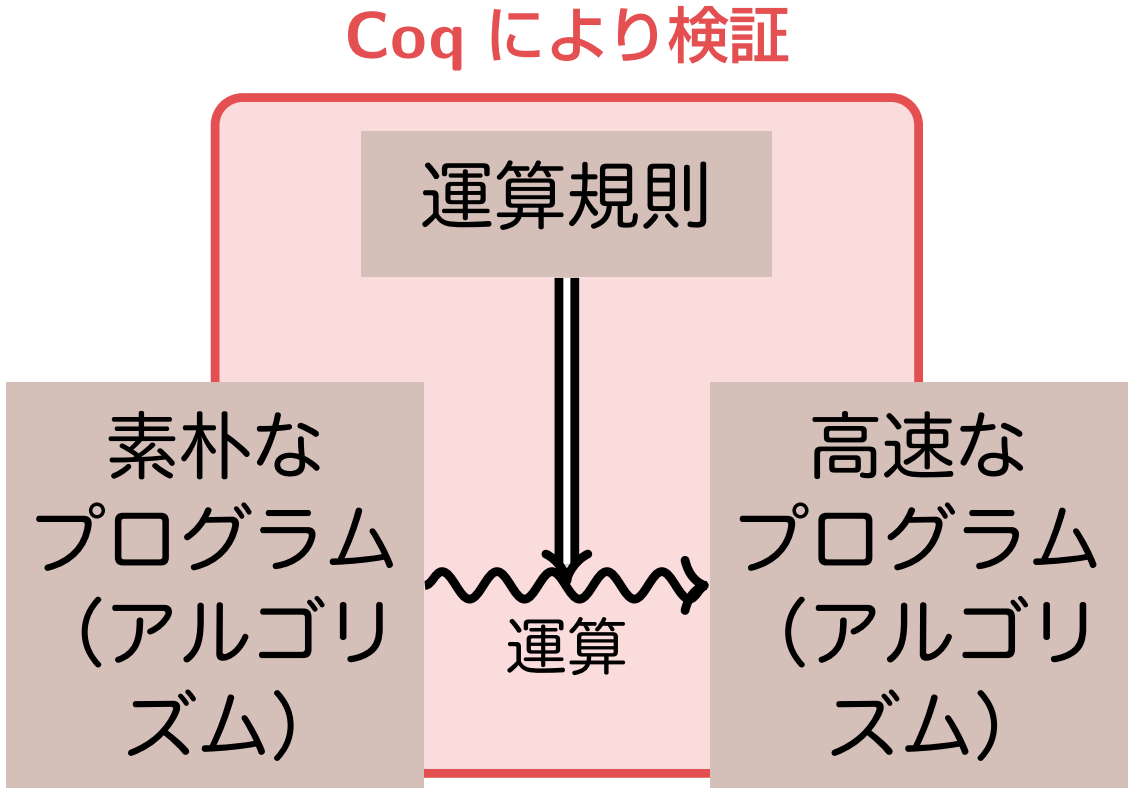
本研究の問題意識

- 運算は本当に正しい？



本研究の問題意識

- 運算は本当に正しい？　→ **Coq で検証！**



map 融合則の Coq による証明例

```
Fixpoint map (X Y : Type) (f : X -> Y) (xs : list X) :=
  match xs with
    [] => [] | (x::xs') => (f x)::(map f xs')
  end.
```

検証の対象とするプログラム定義

map 融合則の Coq による証明例

```
Fixpoint map (X Y : Type) (f : X -> Y) (xs : list X) :=
  match xs with
    [] => [] | (x::xs') => (f x)::(map f xs')
  end.

Theorem map_fusion:
  forall (X Y Z : Type) (f : X -> Y) (g : Y -> Z),
    map g ∘ map f = map (g ∘ f).
```

示したい性質（論理式）

map 融合則の Coq による証明例

```
Fixpoint map (X Y : Type) (f : X -> Y) (xs : list X) :=
  match xs with
    [] => [] | (x::xs') => (f x)::(map f xs')
  end.

Theorem map_fusion:
  forall (X Y Z : Type) (f : X -> Y) (g : Y -> Z),
    map g ∘ map f = map (g ∘ f).

Proof.
  intros ? ? ? ? ?; extensionality xs; cbn.
  induction xs as [|? ? IHx];
  - reflexivity.
  - cbn; rewrite IHx; reflexivity.
Qed.
```

証明

先行研究 [Tesson+, 2011]

- プログラム運算のための Coq ライブラリ
  - 運算で一般的な**等式変形記法による証明を実現**
  - **リストを扱う**プログラムの運算規則を証明
    - Bird, Theory of Lists に基づく

```
Theorem filt_prom : p <| :o: @concat A = @concat A :o: p <| *.
Proof.
    LHS
= { rewrite (filter mapBaseColoruce) }
    (++ / :o: f * :o: @concat A).
= { rewrite map promotion }
    (++ /:o: @concat (list A) :o: f* *).
= { rewrite comp assoc }
(* OMITTED ... *)
Qed.
```

本研究の問題意識:
リストだけでは不十分！

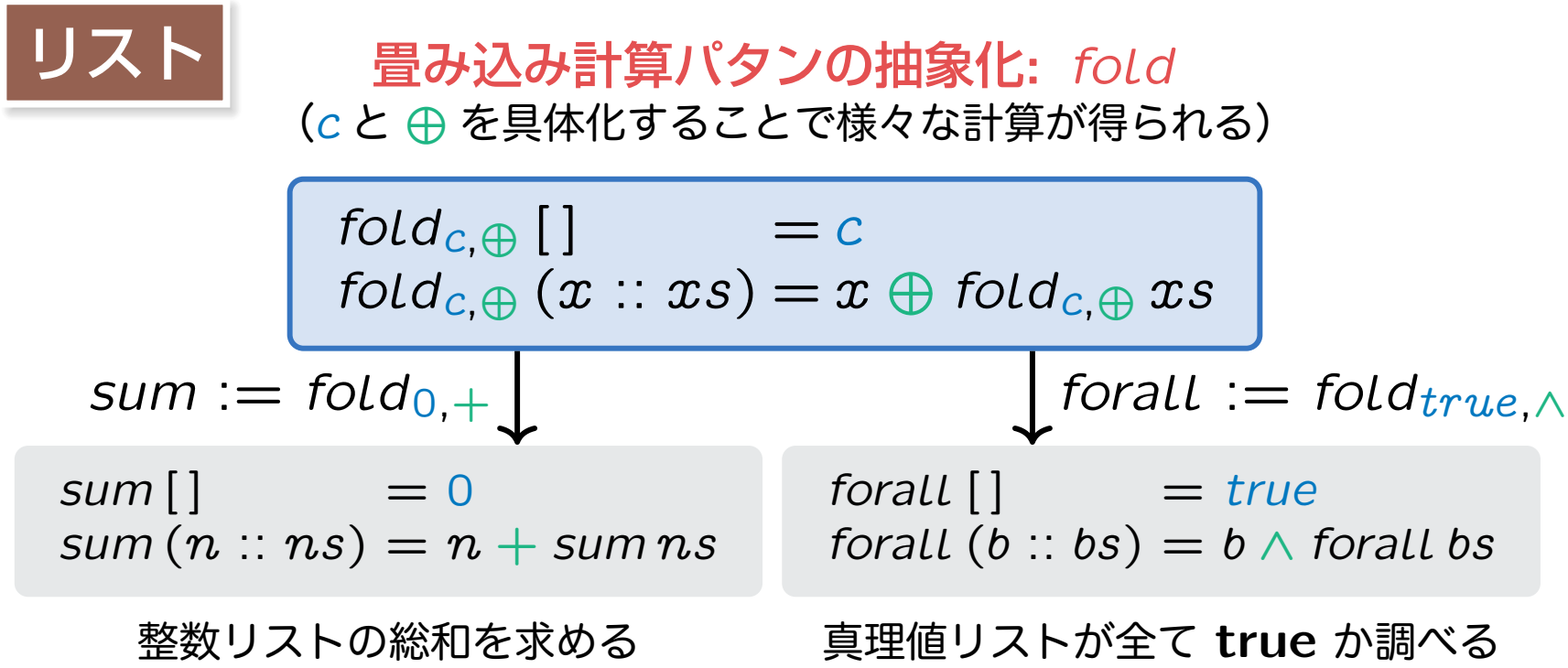
- 関数プログラマは様々なデータ型を用いる
  - 自然数，リスト，二分木，…
  - ユーザ定義 ADT（C でいう構造体のようなもの）
- **リスト以外を扱うプログラムでも運算したい**
  - データ型ごとに運算規則を用意するのは面倒
  - **データ型汎用な運算規則**が望まれる

再帰図式（Recursion Scheme）

再帰図式の説明の前に…

リストの畳み込み計算 *fold*

- リスト上の再帰計算を**高階関数**で抽象化
  - 様々なリスト関数が *fold* として抽象化できる



再帰図式へ一般化:
fold をデータ型汎用に一般化

- さらに様々なデータ型上の fold を関手 F と始 F 代数（終 F 余代数）で抽象化

fold の抽象化: Catamorphism

（関手 F を具体化することで様々なデータ型上の畳み込みが得られる）

$$(\!|\varphi|\!)_F \circ \mathsf{in}_F = \varphi \circ F\,(\!|\varphi|\!)_F$$

リスト

$$\begin{aligned} \mathit{fold}_{c,\oplus}\,[\,] &= c \\ \mathit{fold}_{c,\oplus}\,(x :: xs) &= x \oplus \mathit{fold}_{c,\oplus}\,xs \end{aligned}$$

sum *forall* ...

リストのための fold

二分木

$$\begin{aligned} &\mathit{foldt}_{n,f}\ \mathbf{leaf} = n \\ &\mathit{foldt}_{n,f}\ \mathbf{node}\ t_0\ x\ t_1 \\ &\quad = f\ (\mathit{foldt}_{n,f}\ t_0)\ x\ (\mathit{foldt}_{n,f}\ t_1) \end{aligned}$$

intrav *max* ...

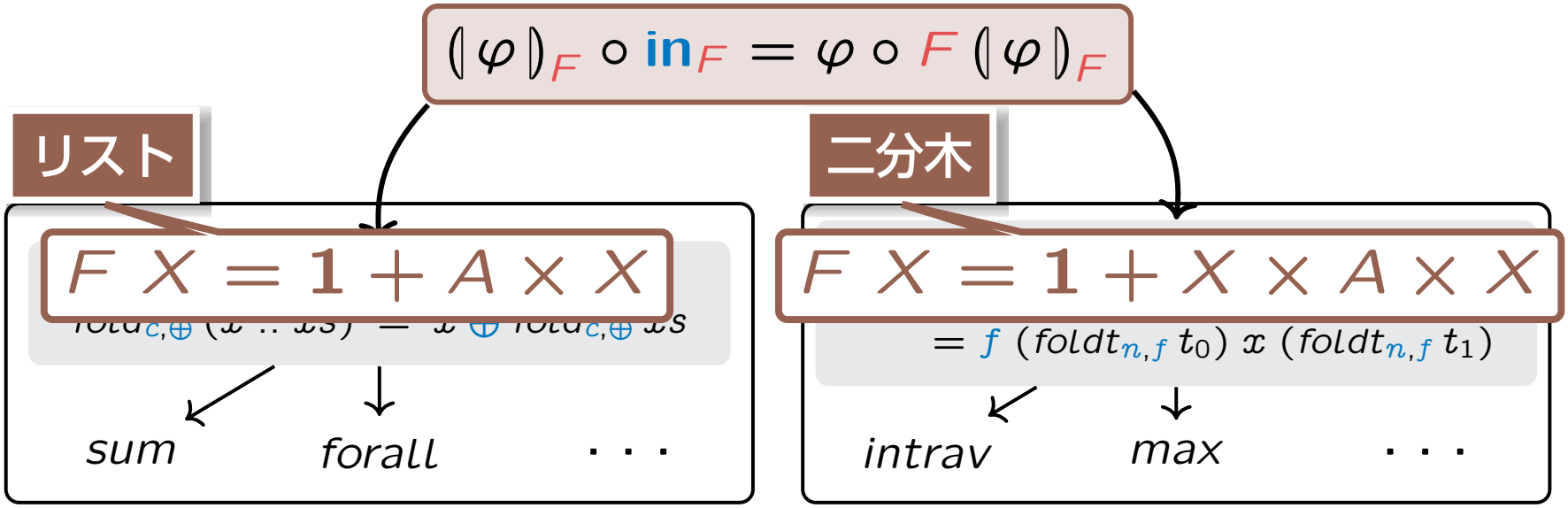
二分木のための fold

再帰図式へ一般化:
fold をデータ型汎用に一般化

- さらに様々なデータ型上の fold を関手 F と始 F 代数（終 F 余代数）で抽象化

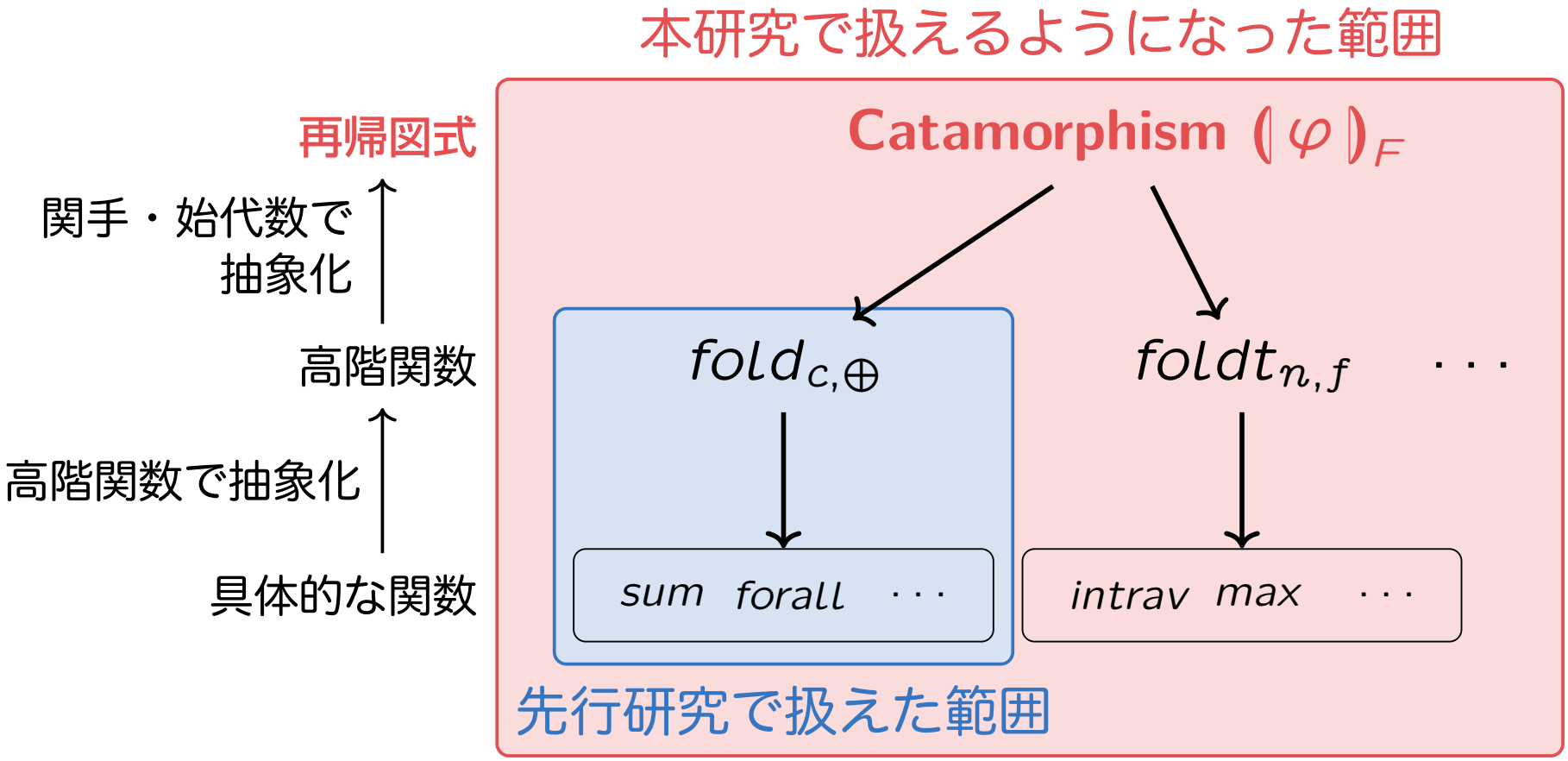
fold の抽象化: Catamorphism

（関手 F を具体化することで様々なデータ型上の畳み込みが得られる）



リストのための fold　　二分木のための fold

ここまでのまとめと本研究の位置付け



データ型汎用な map

- 再帰図式で**データ型汎用な map を実装**できる:

$$\mathbf{map}_F\ f = (\!|\, \mathbf{in}_F \circ F(f, \mathrm{id}) \,|\!)_F$$

  - 関手 F の選び方によって色々なデータ型に対応
- **データ型汎用な map 融合則が記述できる**

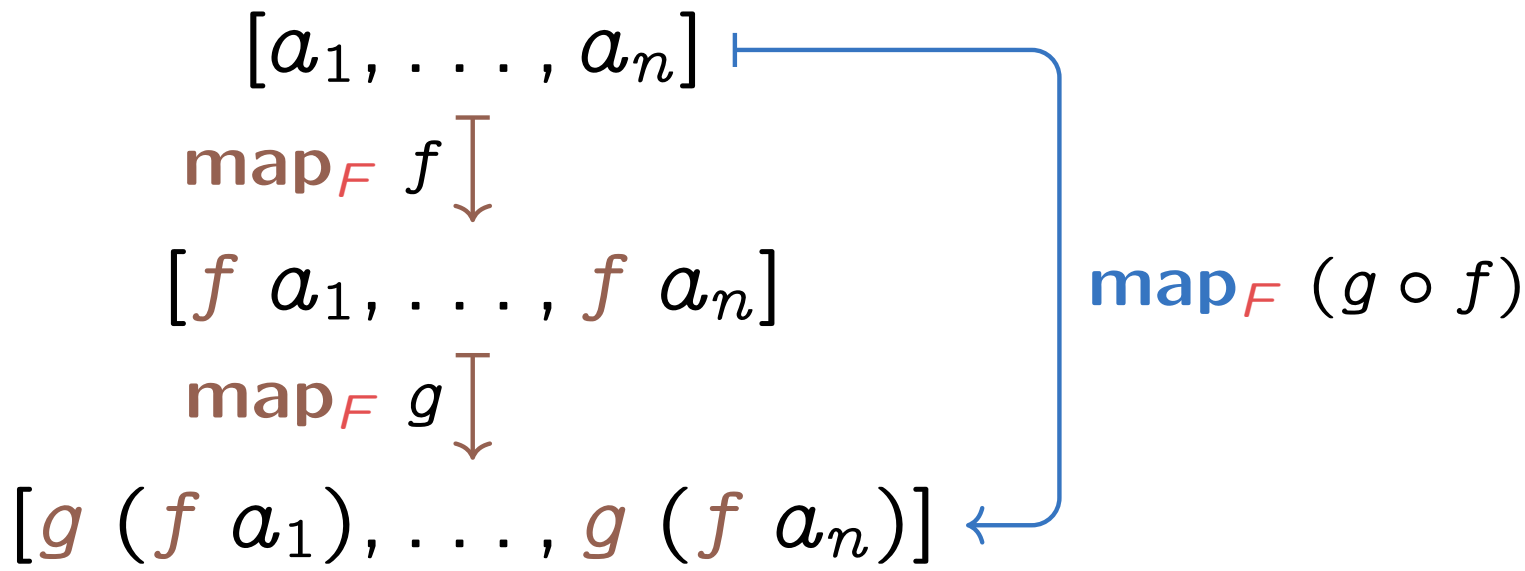
$$\mathbf{map}_F\ g \circ \mathbf{map}_F\ f = \mathbf{map}_F\ (g \circ f)$$

データ型汎用な運算規則の例

一般化 map 融合則

$$(\mathbf{map}_F\ g) \circ (\mathbf{map}_F\ f) = \mathbf{map}_F\ (g \circ f)$$

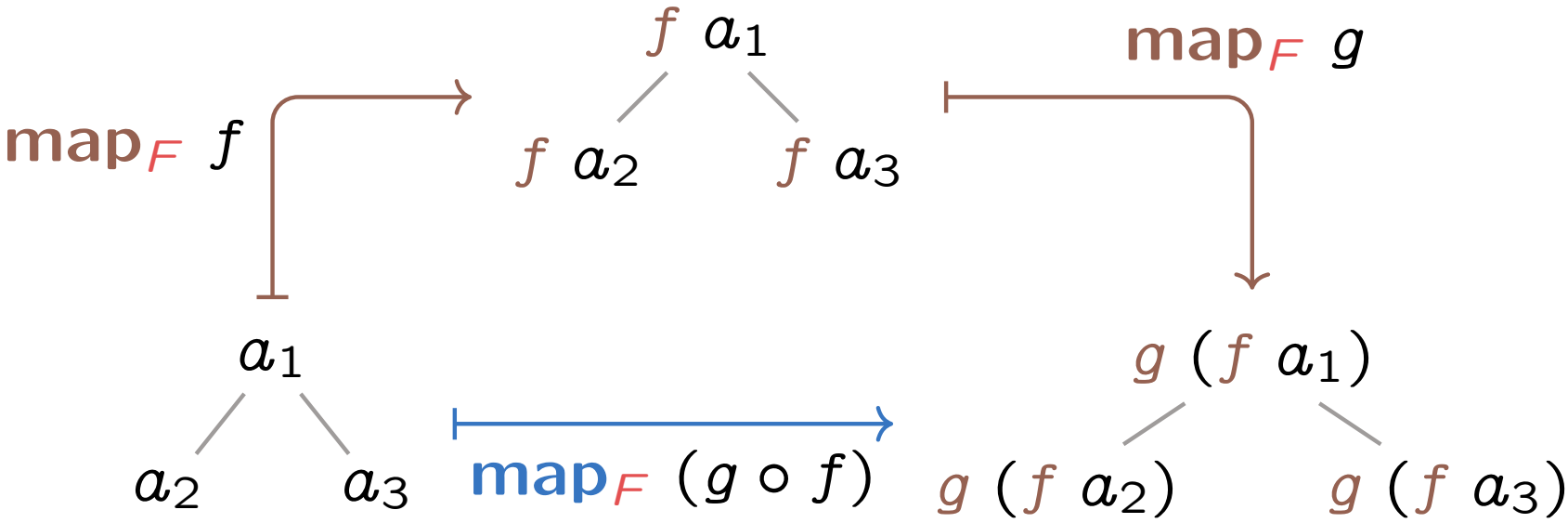
For Lists $(F\ X = \mathbf{1} + A \times X)$



データ型汎用な運算規則の例
一般化 map 融合則

$$(\mathrm{map}_F\ g) \circ (\mathrm{map}_F\ f) = \mathrm{map}_F\ (g \circ f)$$

For Trees $(F\ X = 1 + X \times A \times X)$



本研究: 再帰図式のためのライブラリ

- データ型汎用なプログラム定義や運算定理を扱うための手法を提案
  - 型クラスを用いて関手，始代数などの概念を形式化
- **必要な数学的基礎となる命題群を形式的証明**
  - 関手の数学的性質など
- 以上を Coq ライブラリとして整備
  - Tesson らと同様に，プログラム運算で標準的な方針・記法による形式的証明が可能に

提案ライブラリの評価

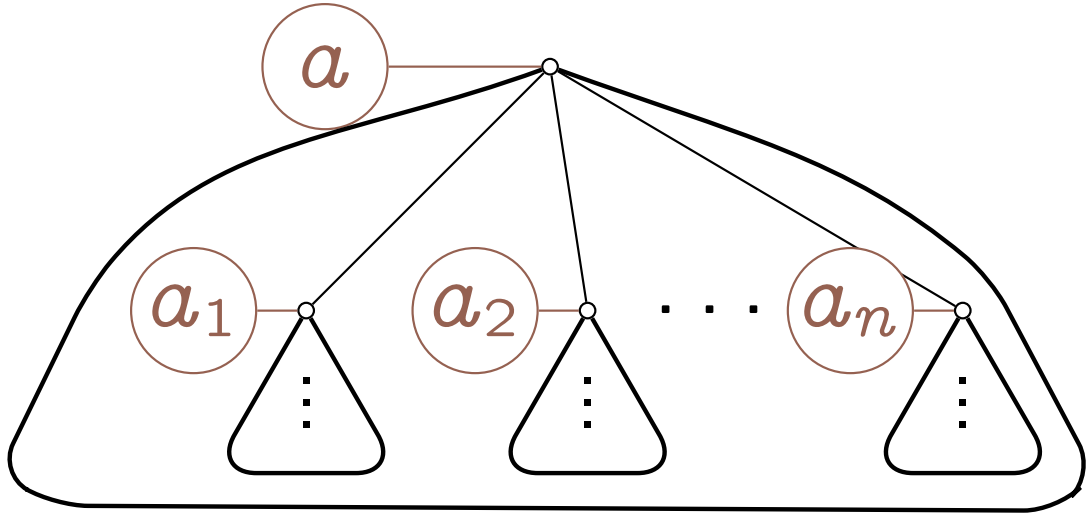
- [Uustalu+ 1999] の定理をすべて形式化が可能であることを確認
  - 基礎的な再帰図式の運算規則が述べられた論文
  - **全 35 定理**を形式化（約 950 行のスクリプト）
- Catamorphism などの再帰図式を用いたプログラム定義や証明が可能に

cata_charn	:	$f \circ \mathtt{in\_} = \varphi \circ F[f] \leftrightarrow f = (\!|\varphi|\!)$
cata_cancel	:	$(\!|\varphi|\!) \circ \mathtt{in\_} = \varphi \circ F[(\!|\varphi|\!)]$
cata_refl	:	$\mathrm{id} = (\!|\mathtt{in\_}|\!)$
cata_fusion	:	$f \circ \varphi = \psi \circ F[f] \to f \circ (\!|\varphi|\!) = (\!|\psi|\!)$
lemma1	:	$\mathtt{in\_} \circ (\!|F[\mathtt{in\_}]|\!) = \mathrm{id} \wedge (\!|F[\mathtt{in\_}] \circ \mathtt{in\_}|\!) = \mathrm{id}$
Def. of in^{-1}		$\mathtt{in\_inv} := (\!|F[\mathtt{in\_}]|\!)$
in_inv_charn	:	$\mathtt{in} \circ \mathtt{in\_inv} = \mathrm{id} \wedge \mathtt{in\_inv} \circ \mathtt{in\_} = \mathrm{id}$
ana_charn	:	$\mathtt{out\_} \circ f = F[f] \circ \varphi \leftrightarrow f = [\!|\varphi|\!]$
ana_cancel	:	$\mathtt{out\_} \circ [\!|\varphi|\!] = F[[\!|\varphi|\!]] \circ \varphi$
ana_refl	:	$[\!|\mathtt{out\_}|\!] = \mathrm{id}$
ana_fusion	:	$\psi \circ f = F[f] \circ \varphi \to [\!|\psi|\!] \circ f = [\!|\varphi|\!]$
Def. of out^{-1}		$\mathtt{out\_inv} := [\!|F[\mathtt{out\_}]|\!]$
out_inv_charn	:	$\mathtt{out\_inv} \circ \mathtt{out\_} = \mathrm{id} \wedge \mathtt{out\_} \circ \mathtt{out\_inv} = \mathrm{id}$
lemma2	:	$f \circ \mathtt{in\_} = \varphi \circ F[\langle f, \mathrm{id}\rangle] \leftrightarrow f = \mathsf{fst} \circ (\!|\langle \varphi, \mathtt{in\_} \circ F[\mathsf{snd}]\rangle|\!)$
Def. of para.		$\langle\!|\varphi|\!\rangle := \mathsf{fst} \circ (\!|\langle \varphi, \mathtt{in\_} \circ F[\mathsf{snd}]\rangle|\!)$
para_charn	:	$f \circ \mathtt{in\_} = \varphi \circ F[\langle f, \mathrm{id}\rangle] \leftrightarrow f = \langle\!|\varphi|\!\rangle$
para_cancel	:	$\langle\!|\varphi|\!\rangle \circ \mathtt{in\_} = \varphi \circ F[\langle \langle\!|\varphi|\!\rangle, \mathrm{id}\rangle]$
para_refl	:	$\mathrm{id} = \langle\!|\mathtt{in\_} \circ F[\mathsf{fst}]|\!\rangle$
para_fusion	:	$f \circ \varphi = \psi \circ F[f \otimes \mathrm{id}] \to f \circ \langle\!|\varphi|\!\rangle = \langle\!|\psi|\!\rangle$
para_cata	:	$(\!|\varphi|\!) = \langle\!|\varphi \circ F[\mathsf{fst}]|\!\rangle$
para_any	:	$f = \langle\!|f \circ \mathtt{in\_} \circ F[\mathsf{snd}]|\!\rangle$
Def. of apo.		$[\!|\varphi|\!] := [\!|[\varphi, F[\mathsf{inr}] \circ \mathtt{out\_}]|\!] \circ \mathsf{inl}$
apo_charn	:	$\mathtt{out\_} \circ f = F[f, \mathrm{id}] \circ \varphi \leftrightarrow f = [\!|\varphi|\!]$
apo_cancel	:	$\mathtt{out\_} \circ [\!|\varphi|\!] = F[[[\!|\varphi|\!], \mathtt{id}]] \circ \varphi$
apo_refl	:	$\mathrm{id} = [\!|F[\mathsf{inl}] \circ \mathtt{out\_}|\!]$
apo_fusion	:	$\psi \circ f = F[f \oplus \mathtt{id}] \circ \varphi \to [\!|\psi|\!] \circ f = [\!|\varphi|\!]$
apo_ana	:	$[\!|\varphi|\!] = [\!|F[\mathsf{inl}] \circ \varphi|\!]$
apo_any	:	$f = [\!|F[\mathsf{inr}] \circ \mathtt{out\_} \circ f|\!]$
lemma3	:	$f \circ \mathtt{in\_} = \varphi \circ F[(\!|\langle f, \mathtt{in\_inv}\rangle|\!)]$ $\leftrightarrow f = \mathsf{fst} \circ \mathtt{out\_} \circ (\!|\mathtt{out\_inv} \circ \langle \varphi, \mathrm{id}\rangle|\!)$
Def. of histo.		$\{\!|\varphi|\!\} := \mathsf{fst} \circ \mathtt{out\_} \circ (\!|\mathtt{out\_inv} \circ \langle \varphi, \mathrm{id}\rangle|\!)$
histo_charn	:	$f \circ \mathtt{in\_} = \varphi \circ F[[\!|\langle f, \mathtt{in\_inv}\rangle|\!]] \leftrightarrow f = \{\!|\varphi|\!\}$
histo_cancel	:	$\{\!|\varphi|\!\} \circ \mathtt{in\_} = \varphi \circ F[[\!|\langle \{\!|\varphi|\!\}, \mathtt{in\_inv}\rangle|\!]]$
histo_refl	:	$\mathrm{id} = \{\!|\mathtt{in\_} \circ F[\mathsf{fst} \circ \mathtt{out\_}]|\!\}$
histo_fusion	:	$f \circ \varphi = \psi \circ F[[\!|(f \otimes \mathrm{id}) \circ \mathtt{out\_}|\!]] \to f \circ \{\!|\varphi|\!\} = \{\!|\psi|\!\}$
histo_cata	:	$(\!|\varphi|\!) = \{\!|\varphi \circ F[(\mathsf{fst} \circ \mathtt{out\_})]|\!\}$
Def. of futu.		$[\!|\varphi|\!] := [\!|[\varphi, \mathrm{id}] \circ \mathtt{in\_inv}|\!] \circ \mathtt{in\_} \circ \mathsf{inl}$
futu_charn	:	$\mathtt{out\_} \circ f = F[(|f, \mathtt{out\_inv}|)] \circ \varphi \leftrightarrow f = [\!|\varphi|\!]$
futu_cancel	:	$\mathtt{out\_} \circ [\!|\varphi|\!] = F[(|[[\!|\varphi|\!], \mathtt{out\_inv}|)]) \circ \varphi$
futu_refl	:	$\mathrm{id} = [\!|F[\mathtt{in\_} \circ \mathsf{inl}] \circ \mathtt{out\_}|\!]$
futu_fusion	:	$\psi \circ f = F[(|\mathtt{in\_} \circ (f \oplus \mathrm{id})|)] \circ \varphi \to [\!|\psi|\!] \circ f = [\!|\varphi|\!]$
futu_ana	:	$[\!|\varphi|\!] = [\!|F[(\mathtt{in\_} \circ \mathsf{inl}) \circ \varphi]|\!]$

```
Theorem data_functor :
  forall {A B C : Type} (f : A -> B) (g : B -> C),
    (fmap g) ∘ (fmap f) = fmap (g ∘ f).
Proof.
  intros; unfold fmap.
  assert ( (fmap g) ∘ in_ ∘ F⟨f⟩[id] = in_ ∘ F⟨g ∘ f⟩[id] ∘ F⟨id⟩[fmap g] ).
  {
    unfold fmap.
      Left
    = ( in_ ∘ (F⟨g⟩[id] ∘ F⟨id⟩[(| in_ ∘ F⟨g⟩[id] |)]) ∘ F⟨f⟩[id] )
        { by cata_cancel }.
    = ( in_ ∘ (F⟨g⟩[(| in_ ∘ F⟨g⟩[id] |)] ∘ F⟨f⟩[id]) )
        { by fmap_functor_dist }.
    = ( in_ ∘ (F⟨g ∘ f⟩[(| in_ ∘ F⟨g⟩[id] |)]) )    { by fmap_functor_dist }.
    = ( in_ ∘ (F⟨g ∘ f⟩[id] ∘ F⟨id⟩[(| in_ ∘ F⟨g⟩[id] |)]) )
        { by fmap_functor_dist }.
    = Right.
  }
  unfold fmap in H0.
  Left
  = ( (| in_ ∘ F⟨g⟩[id] |) ∘ (| in_ ∘ F⟨f⟩[id] |) ).
  = ( (|in_ ∘ F ⟨g ∘ f⟩[id] |) )  { apply cata_fusion }.
  = Right.
Qed.
```

なぜ [Uustalu+, 1999] なのか？

- Histomorphism の提案論文
- Histomorphism は有用：**DP ができる！**
  - Histo: Cata + **過去の計算履歴へのアクセス**
    - 中間データ構造: 余自由余モナド



畳み込み

(帰納的データ型を消費)

Catamorphism[Malcom, 1990]

$$f = (\!|\varphi|\!)_F \Leftrightarrow f = \varphi \circ F(f) \circ \mathbf{in}_F^{-1}$$

Paramorphism[Meertens, 1992]

$$\langle\!|\varphi|\!\rangle_F \Leftrightarrow f = \varphi \circ F(\langle f, \mathrm{id}_{\mu F}\rangle) \circ \mathbf{in}_F^{-1}$$

Histomorphism[Uustalu+, 1999]

$$\{\!|\varphi|\!\}_F \Leftrightarrow f = \varphi \circ F([\![\langle f, \mathbf{in}_F^{-1}\rangle]\!]) \circ \mathbf{in}_F^{-1}$$

反畳み込み

(余帰納的データ型を構築)

Anamorphism[Fokkinga+, 1991]

$$f = [\![\psi]\!]_F \Leftrightarrow f = \mathbf{out}_F^{-1} \circ F(f) \circ \psi$$

Apomorphism[Vos, 1995]

$$f = [\![\varphi]\!]_F \Leftrightarrow f = \mathbf{out}_F^{-1} \circ F([f, \mathrm{id}_{\nu F}]) \circ \psi$$

Futumorphism[Uustalu+, 1999]

$$f = [\![\psi]\!]_F \Leftrightarrow f = \mathbf{out}_F^{-1} \circ F((\![[f, \mathbf{out}_F^{-1}]]\!)) \circ \psi$$

再畳み込み

中間データ構造を構築し，それを消費

Hylomorphism[Meijer+, 1991]

$$f = [\![\varphi, \psi]\!]_F \Leftrightarrow f = \varphi \circ F\, f \circ \psi$$

まとめ

- データ型汎用な運算規則を証明するための Coq ライブラリを提案
  - 関手，始代数などの概念を形式化
- 論文 [Uustalu+, 1999] に現れる全ての定理・定義を形式化
  - データ型汎用な証明が自然に記述できた
- 今後の課題：他の運算規則の形式化を通じた有用性検証
- ここまでは，主に発表論文（Murata et al., APLAS '19；査読付き国際会議）の内容に基づく

ここまでに扱えなかった話題 1:

Coq による証明の記法の改善

- プログラム運算の論文では，等式や様々な二項関係を鎖状に繋げる証明記法が見られる
- 同様の記法を Coq で実現するライブラリを提案
  - 自然数上の不等式の証明へ応用
- （村田ら, 2018；日本語論文誌 PRO）へ掲載

```
Right
=  (ack (Sx) (ack (S (S x)) y)).
>= (S (S (ack (S (S x)) y)))        {by IHx}.
>= (S (S (S (S y))))                {by IHy}.
>= (S (S (S y))) { omega }.
=  Left.
```

畳み込み

(帰納的データ型を消費)

Catamorphism[Malcom, 1990]

$$f = (\!|\varphi|\!)_F \Leftrightarrow f = \varphi \circ F(f) \circ \mathbf{in}_F^{-1}$$

Paramorphism[Meertens, 1992]

$$\langle\!|\varphi|\!\rangle_F \Leftrightarrow f = \varphi \circ F(\langle f, \mathrm{id}_{\mu F}\rangle) \circ \mathbf{in}_F^{-1}$$

Histomorphism[Uustalu+, 1999]

$$\{\!|\varphi|\!\}_F \Leftrightarrow f = \varphi \circ F([\![\langle f, \mathbf{in}_F^{-1}\rangle]\!]) \circ \mathbf{in}_F^{-1}$$

反畳み込み

(余帰納的データ型を構築)

Anamorphism[Fokkinga+, 1991]

$$f = [\![\psi]\!]_F \Leftrightarrow f = \mathbf{out}_F^{-1} \circ F(f) \circ \psi$$

Apomorphism[Vos, 1995]

$$f = [\![\varphi]\!]_F \Leftrightarrow f = \mathbf{out}_F^{-1} \circ F([f, \mathrm{id}_{\nu F}]) \circ \psi$$

Futumorphism[Uustalu+, 1999]

$$f = [\![\psi]\!]_F \Leftrightarrow f = \mathbf{out}_F^{-1} \circ F((\![[f, \mathbf{out}_F^{-1}]]\!)) \circ \psi$$

再畳み込み

中間データ構造を構築し，それを消費

Hylomorphism[Meijer+, 1991]

$$f = [\![\varphi, \psi]\!]_F \Leftrightarrow f = \varphi \circ F\, f \circ \psi$$

ここまでに扱えなかった話題 2:

再畳み込み再帰図式の形式化に向けて

- 再畳み込み再帰図式は重要
  - 酸性雨定理 [Takano, '95] などの強力な運算規則
- Coq で hylomorphism を扱うのは素朴には難しい
  - 全域関数の圏 **Set** では定義できないから
  - 領域理論的な部分関数に関わる数学的基礎が必要
- Delay モナド + monadic recusion scheme を利用し，Delay モナド上の Kleisli 圏 $\mathbf{Kle}_{\mathbf{D}}\ \mathbf{Set}$ 上で再帰図式を理論的整理
- （PPL 2020；国内ワークショップ）に採択

「定理証明支援系 Coq によるデータ型汎用なプログラム運算」

Keywords: 関数型言語，Coq，再帰図式，catamorphism，histomorphism，[Uustalu+, 1999]，データ型汎用，プログラム運算，圏論

- 一般的な polymorphism（多相型）とこの発表の「データ型汎用」の違いは何か？（少し解答が長くなります）
- 色々な種類の再帰図式があったが，それぞれ何が違う？
- Coq 自体の正しさはどう保証するのか？
- 具体的に運算でどれぐらい計算量が落ちるのか？
- 提案手法の具体的な応用例はあるか？（何に使える？）
- 運算の自動化は難しいのか？
- そもそもプログラム運算なんて誰がやるのか？
  （世の中に圏論使ったプログラミングができる人がどれだけいるのか？）
- そもそも Coq というのは主に誰が使っているのか？
- 始代数のための型クラスって具体的に？
- 提案ライブラリは結局誰が使うのか？